High-level modeling of software-management interactions and tasks for autonomic computing*

Edin Arnautovic, Hermann Kaindl, Jürgen Falb, Roman Popp
Institute of Computer Technology
Vienna University of Technology, Vienna, Austria
{arnautovic, kaindl, falb, popp}@ict.tuwien.ac.at

Abstract

For making software systems autonomic, it is important to understand and model software-management tasks. Each such task contains typically many interactions between the administrator and the software system.

We propose to model software-management interactions and tasks in the form of a discourse between the administrator and the software system. Such discourse models are based on insights from theories of human communication. This should make them "natural" for humans to define and understand. While it may be obvious that such discourse models cover software-management interactions*, we found that they may also represent major parts of the related* tasks*. Our well-defined models of interactions and tasks as well as their operationalization should facilitate their execution and automation.*

1 Introduction

Today's software systems are usually distributed and very complex, having a large amount of parameters and possible configurations. It is crucial to satisfy quality requirements of these systems such as performance, availability and security. Management of these systems includes tasks required to control, measure, optimize, troubleshoot and configure software in a computing system. These management tasks are nowadays performed by well-trained professionals which are responsible for *configuring* the system so that the users can get their jobs done and for *maintaining* the system against both internal failures and internal or external attacks [1]. They interact with the system using command-line interfaces, graphical interfaces or web-based management tools [5].

*This research has been carried out in the *OntoUCP* project, partially funded by the FIT-IT Program of the Austrian FFG and Siemens AG Österreich as project number 809254/9312.

In order to automate software systems management tasks, it is important to understand and represent them in some more or less formal way. Any of such tasks contains typically many interactions between the administrator and the managed software system. We believe that modeling of these interactions facilitates understanding and specifying tasks as well. In order to make interactions easy to understand and to specify by humans, their specification should be on a high level.

Thus, we propose to model the software systems' management tasks in the form of discourses between the administrator and the system. In our approach, such discourse models are based on insights from human communication theories and provide specifications for tasks and their interactions.

The remainder of this paper is organized in the following manner. First, we provide some background on the human communication theories used. Then we illustrate the discourse model based interaction and task specifications in our approach, as well as the high-level autonomic architecture. Finally, we sketch procedural semantics of our specifications and discuss related work.

2 Human Communication Theories

Both tasks and their interactions can be specified in many ways. We strive for a uniform high-level approach to task and interaction representation based on the following human communication theories.

Communicative acts are derived from speech acts [19] and represent basic units of language communication. Thus, any communication can be seen as enacting of communicative acts, acts such as making statements, giving commands, asking questions and so on. *Communicative Acts* carry the intention of the interaction (e.g., asking a *Question*) and can be further classified into *Assertions*, *Directives* and *Commissives*. *Assertions* convey information without requiring receivers to act beside changing their beliefs (e.g., In-

0-7695-3093-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAS.2008.46

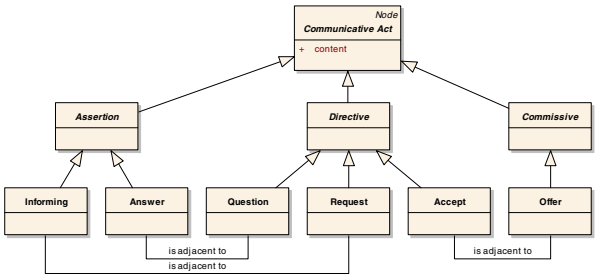

Figure 1. Part of Communicative Acts Hierarchy.

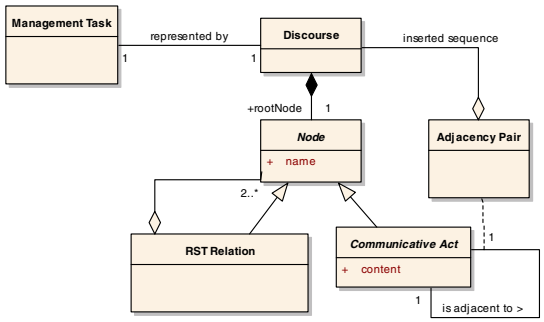

Figure 2. Interaction and Task Specification Metamodel.

forming and Answer). *Directives* (e.g., Question, Request, Accept) and *Commissives* (e.g., Offer) require an action by the receiver or sender for the advancement of the dialog by further communicative acts. This classification is shown in Figure 1. The figure shows only a small selection from many communicative acts. Communicative acts have been successfully used in several applications: inter-agent communication in FIPA Agent Communication Language[1] (ACL), information systems [18] and high-level specifications of user interfaces [7].

Conversation Analysis. While communicative acts are useful concepts to account for intention in an isolated utterance, representing the relationship between utterances needs further theoretical devices. We have found inspiration in Conversation Analysis [16] for this purpose. Conversation analysis focuses on sequences of naturally-occurring talk "turns" to detect patterns that are specific to human oral communication, and such patterns can be regarded as familiar to the user. In our work we make use of patterns such as "adjacency pair" and "inserted sequence".

Rhetorical Structure Theory (RST) [17] is a linguistic theory focusing on the function of text, widely applied to the automated generation of natural language. It describes internal relationships among text portions and associated constraints and effects. The relationships in a text are organized in a tree structure, where the rhetorical relations are associated with non-leaf nodes, and text portions with leaf nodes. In our work we make use of RST for linking communicative acts and further structures made up of RST relations. Thus, they represent the structure of possible interactions between an administrator and the software system. We use two types of RST relations: symmetric, multi-nuclear (e.g., *Joint, Otherwise*) and asymmetric, nucleus-satellite (e.g., *Result, Condition, Elaboration*).

[1]Foundation for Intelligent Physical Agents, Communicative Act Library Specification, www.fipa.org

3 Interaction and Task Specification

We have developed a metamodel which defines what the structure of the interaction and task models should look like in our approach. We explain it here using the UML class diagram[2] in Figure 2 and through management examples shown in Figures 4 and 5. Both example management tasks take place within a typical three-tier architecture as shown in Figure 3.

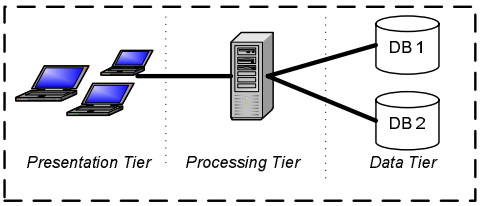

Figure 3. Example Architecture.

The metamodel in Figure 2 shows that a Management Task is represented by a discourse, which contains communicative acts, adjacency pairs and RST relations. A *Management Task* represents a typical task of software system management such as system optimization, recovering from errors, etc. It is specified in terms of interactions which occur in a *Discourse*. A single interaction is represented with a *Communicative Act* where its type specifies the intention of the utterance like *requesting* information about a system's status. *RST Relations* relate *Node*s which can be communicative acts or other RST relations, thus building up the hierarchical structure of the discourse.

The example in Figure 4 represents the management task of optimizing the system in the case of a too long response time, and is specified as a discourse which takes place between the administrator and the managed system. The struc-

[2]At the time of this writing, the specification of UML is available at `http://www.omg.org`.

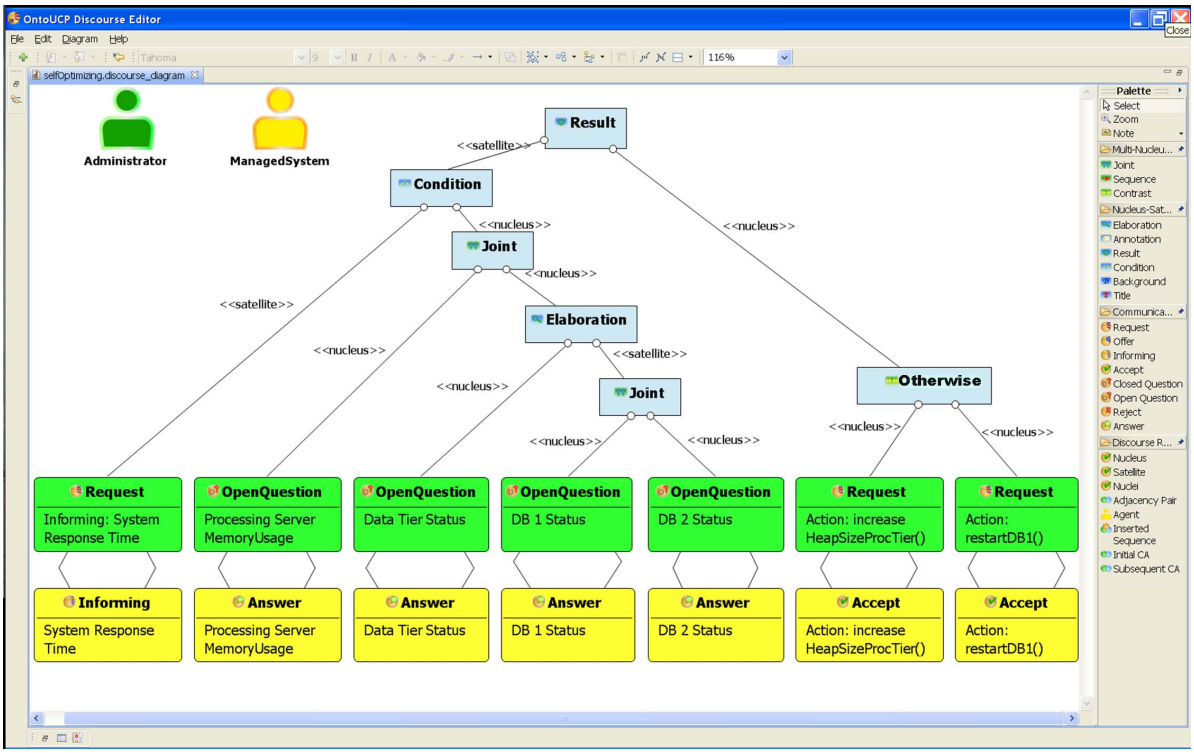

Figure 4. System Optimizing Task Specification.

ture of this specification is composed from RST relations (shown in boxes) and the Communicative Acts (shown in rounded boxes). Each of the Communicative Acts belongs to either the Administrator or the ManagedSystem. For an explanation, let us start at the top with the relation called *Result*. It represents that the actions — requesting the increase of the heap size of the Processing Tier *or*[3] requesting the restart of the DB1 — are a consequence of the situation resulting from the preceding interactions. In the course of the interactions, some require a certain condition to be fulfilled. E.g., the *Condition* relation requires information about the current Server Response Time and the execution continues only if its value is beyond some defined limit. The acquisition of information about the status of the Data Tier can be further *elaborated*, adding more details by asking *Questions* about the status of each of two databases running within the Data Tier.

Such a tree of RST relations could be viewed as the design rationale of the interactions. While the discourse model represents a generic set of possible discourses, the concrete interaction flow will be controlled by the Decision Making component within the Autonomic Manager (Figure 6). In a typical case of software management, the interaction between an administrator and the system is not symmetric. Obviously, the administrator guides the discourse and the system only acts on her behalf and responds. The presented example serves for illustration purposes only. A real-life scenario would involve more interactions (e.g., Questions about the Presentation Tier or more details about the Processing Tier). However, this would clutter our diagram here.

In some cases, however, the managed system can take the initiative and inform the administrator about some situation in the system. A model of such a discourse is shown in our second example in Figure 5. In this case, the administrator registers (left in the figure: *Request* (Action: ...) – *Accept* (Registration)) for information about errors in the system. Registration is a *Condition* for the further discourse flow and the system sends the *Informing* Communicative Act when some ominous situation occurs. This can *result* in redeploying App 1 (on the right side in Figure 5). While the registration process takes place only once, the information about the error and resulting corrective actions can happen several times.

It is important to note that we put our emphasis on the intention in the communication (e.g., is a particular interaction a *Question*, *Informing* or a *Request*) and abstract from the means of the communication. E.g., in the first example the interactions can take place using graphical or command-

[3]The "or" is modeled using the RST Relation *Otherwise*.

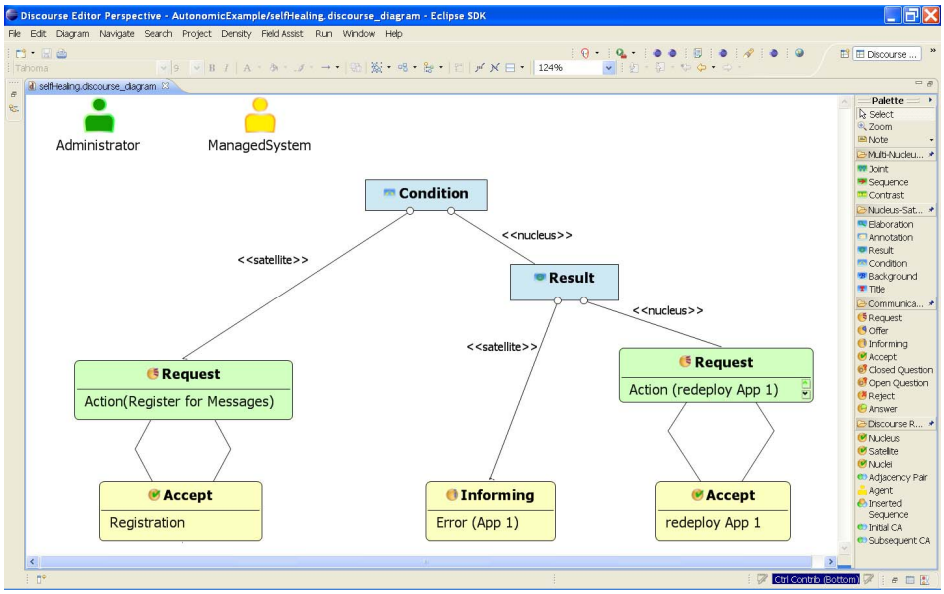

Figure 5. System Healing Task Specification.

line interfaces and in the second the events can be displayed on the screen but also sent via emails or textual phone messages. The discourse model for the interaction and task specification would still be the same.

Such discourse models cover software-management interactions and represent related tasks. Since our discourse models are based on human communication theories, they should be easier to analyze and design than traditional approaches to communication specification.

4 Autonomic Architecture

Figure 6 illustrates a sketch of our proposed autonomic architecture designed to execute and automate modeled tasks as presented above. It is based on the generic autonomic architecture [9] separating the autonomic manager from the managed system.

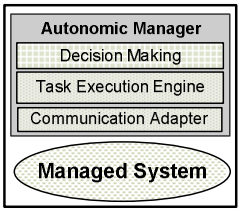

Figure 6. Autonomic Architecture.

The *Task Execution Engine* interprets the task (more precisely its statechart representation, as defined below) and manages the flow of the communication. The *Decision Making Component* controls the whole autonomic administration process by deciding which next set of actions (set of communicative acts) has to be performed. The Decision Making component can be more or less complex, e.g., it can simply interpret some predefined rules or be implemented as an expert system. The *Communication Adapter* encapsulates low-level interaction interfaces of the managed system. Details of these components are beyond the scope of this paper.

5 Procedural Semantics

An important issue is to operationalize our interaction and task specifications and to make them executable within the Task Execution Engine. In order to achieve this operationalization, we transform our interaction and task specifications in the form of discourses into state machines, where the generation of the program code for the state machine is based on the well-known State Design Pattern [8]. This transformation to state machines defines procedural semantics of our discourse models.

In many real-world applications, predetermined discourse processes are well suitable for communication between human and computer, since the user can anticipate the behavior of the system and can expect the same user interfaces whenever she starts the discourse with the system again. This behavior can be achieved by using state machines for the discourse management. Since, especially in user interfaces the possible interactions are manifold, a flat

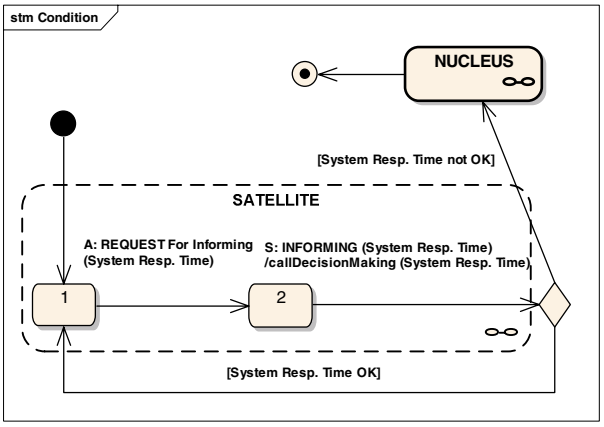

Figure 7. Mapping of a *Condition* RST Relation

state machine can become quite complicated. A solution for reducing the complexity of state machines are statecharts, which introduce hierarchies into state machines. Thus, we use statecharts to specify the procedural semantics of our interaction and task specifications. The statecharts derived from the interaction and task specification specify the dynamics of the complete discourse of a task.

Such statecharts have the following basic structure:

- Each state transition corresponds to exactly one communicative act and thus represents the advancement in the dialog.
- State transitions are triggered by sending a communicative act by either the administrator or by the managed system.
- The Task Execution Engine processes each state, resulting in system effects and in possible triggering of a new communicative act.
- Adjacency pairs are reflected in the statechart by a sequence of transitions. Thus, they are constraining the set of potential communicative acts of outgoing transitions of the current and adjacent states.
- Rhetorical relations are mapped to state machine patterns forming a submachine state that can be included in higher-level rhetorical relation mappings.

Let us now explain the mapping of a discourse model to a statechart. It is done by traversing the tree structure and recursively applying statechart mappings of the corresponding RST relations. Typically, the statechart of one RST relation is included as a submachine state in the statechart of the higher level statechart. Therefore, the hierarchy of the overall statechart corresponds to the hierarchy of the discourse

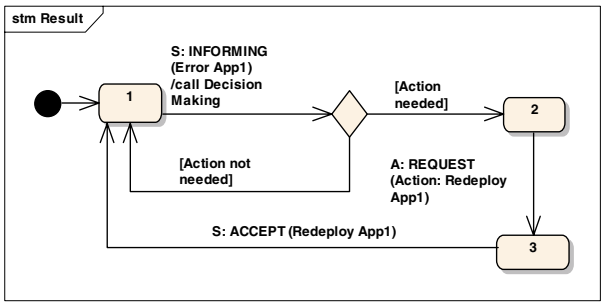

Figure 8. Mapping of a *Result* RST Relation

tree. In the following, we explain the mapping of two of the RST relations to their statecharts using our examples.

The statechart of a *Condition* relation (from the example in Figure 4) is shown in Figure 7. The Condition relation in this case requires that the request issued by the administrator (*A: REQUEST for Informing (System Resp. Time)*) is responded by the system (*S: INFORMING(System Resp. Time)*)[4]. After the response, the Task Execution Engine forwards this information to the Decision Making component (*callDecisionMaking(System Resp. Time)*). The Decision Making Component decides about further processing (modeled by the UML "choice" construct — graphically represented by the diamond). If the response time is above some defined value (*[System Resp. Time not OK]*), the Condition is "fulfilled" and the execution continues with the NUCLEUS submachine state, which is a placeholder for the statechart of the discourse subtree in the nucleus branch of the Condition relation. If not, the statechart goes into the initial state. We can get to the general statechart pattern of the Condition relation by replacing states 1 and 2 — which represent the statechart for the request-informing adjacency pair — with a SATELLITE submachine state and by replacing the concrete transition triggers.

For the *Result* relation we use the second example from Figure 5 for explanation. After the Task Execution Engine receives the Informing about an error, it contacts the Decision Making component, which again decides about the continuation of the discourse. If the action is needed, the Communicative Act *Request(Action: Redeploy App1)* is issued and sent via the Communication Adapter to the managed system, see Figure 8.

6 Related Work

Our work relates both to the field of interaction modeling (between humans and computers as well as between computers) and to the field of autonomic and self-managed software systems.

[4] **A:** represents the Administrator; **S:** represents the Managed**S**ystem

Modeling interaction design is mostly done through techniques from task analysis and cognitive science. Techniques based on Hierarchical Task Analysis [14] or GOMS [10] model activities on various levels of detail in a hierarchical way to achieve a particular goal, and (e.g., temporal) relationships between tasks on the same level. On the more detailed levels, task models specify only the type of tasks (e.g., user, system or interaction task) or operators (click, select ...), but not their intention in the sense of asking, requesting, etc.

Formal interaction modeling is important for interactions between agents. Most approaches for modeling inter-agent communication utilize some finite-state machine. E.g., Labrou and Finin [13] deal with interactions between agents based on KQML, where *conversation policies* are proposed for the description of conversations between agents. Conversation policies represent simple conversations between agents in terms of possible sequences of KQML messages. Our discourse models can represent more complex interactions and should be easier to design by humans.

Modeling and specifying of management tasks and user interfaces for performing those tasks has been neglected in general [4]. However, operators and administrators of software systems are constantly trying to automate administrative tasks and to reduce unnecessary interactions with the system. They use their own executable scripts to automate monitoring of system health, to perform operations on a large number of systems, and to try to eliminate errors on common tasks that take many steps [11]. Our approach also strives for the automation of administrative tasks but concentrates on interactions within such tasks and provides a well-defined way for their modeling.

A typical approach to define automation of software management tasks for autonomic computing is in terms of *policies*. Policies represent instructions to determine the most appropriate activity in a given situation. One way to specify policies has been defined in [3]. They represent policies in the form "`IF condition THEN action`" where *condition* contains a particular state of the system and the *action* represents the actions to be performed if such a state occurs. They also define how to manage and execute such policies. Kephart and Walsh [12] define three types of policies: Action, Goal and Utility Function policies. Action policies are on the lowest level and take also the `if-then` form. Goal policies specify a single desired state and the system should generate behavior itself from the policy. Utility Function policies generalize Goal policies where a desired state is computed by selecting from the present collection of feasible states the one that has the highest utility. The Accord framework [15] defines so-called operational interfaces. This offers the possibility to formulate, inject, and manage rules that are used to manage the runtime behaviors of the autonomic system. Rules incorporate also typical if–then expressions, i.e., "IF condition THEN actions". Similarly to our approach, Cheng et al [6] consider that "the capturing and representation of human expertise in a form executable by a computer" crucial for the automation of management tasks. They have developed a new language for adaptation where the concepts used in the language are derived from system administration tasks. The basic concept in the language is a *tactic*, which embodies a small sequence of actions to fix a specific problem in a localized part of the system. A tactic contains the conditions of applicability, a sequence of actions, and a set of intended effects after execution.

Contrary to this work on policies, our approach focuses on and formalizes interactions between an administrator and a software system. We believe that the interactions are important both for the task execution and for understanding the task. It seems also non-trivial to reduce sometimes complex management tasks to single policies. We believe that our task models should be easier to create by humans since they are based on human communication theories. To facilitate modeling of tasks and their corresponding discourses, we have developed a graphical modeling tool, as presented in the examples above.

Finally, we proposed to utilize similar models as presented in this paper for the transition towards autonomic systems [2].

7 Conclusion

In essence, we propose to model software-management interactions and tasks in the form of a discourse between the administrator and the software system. We believe that this formalization would ease the automation of such interactions and thus the automation of the management tasks related to these interactions. The operationalization of such discourse models enables their execution and automation.

References

[1] E. A. Anderson. *Researching system administration*. PhD thesis, University of California, Berkeley, 2002.

[2] E. Arnautovic, H. Kaindl, J. Falb, R. Popp, and A. Szép. Gradual Transition towards Autonomic Software Systems based on High-level Communication Specification. In *SAC '07: Proceedings of the 2007 ACM Symposium on Applied Computing, Autonomic Computing Track*, pages 84–89, New York, NY, USA, 2007. ACM Press.

[3] R. M. Bahati, M. A. Bauer, and E. M. Vieira. Mapping Policies into Autonomic Management Actions. In *ICAS '06: Proceedings of the International Conference on Autonomic and Autonomous Systems*, page 38, Washington, DC, USA, 2006. IEEE Computer Society.

[4] R. Barrett, Y.-Y. M. Chen, and P. P. Maglio. System administrators are users, too: designing workspaces for manag-

ing internet-scale systems. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 1068–1069, New York, NY, USA, 2003. ACM Press.

[5] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 388–395, New York, NY, USA, 2004. ACM Press.

[6] S.-W. Cheng, D. Garlan, and B. Schmerl. Architecture-based Self-adaptation in the Presence of Multiple Objectives. In *ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Shanghai, China, 21-22 May 2006.

[7] J. Falb, R. Popp, T. Röck, H. Jelinek, E. Arnautovic, and H. Kaindl. Using Communicative Acts in High-Level Specifications of User Interfaces for Their Automated Synthesis. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, pages 429–430, New York, NY, USA, 2005. ACM Press. Tool demo paper.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[9] IBM Corporation. *An architectural blueprint for autonomic computing*, third edition, June 2005. White Paper.

[10] B. E. John and D. E. Kieras. Using GOMS for User Interface Design and Evaluation: Which Technique? *ACM Trans. Comput.-Hum. Interact.*, 3(4):287–319, 1996.

[11] E. Kandogan and J. Bailey. Usable Autonomic Computing Systems: The Administrator's Perspective. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing (ICAC'04)*, pages 18–26, Washington, DC, USA, 2004. IEEE Computer Society.

[12] J. O. Kephart and W. E. Walsh. An Artificial Intelligence Perspective on Autonomic Computing Policies. In *POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, page 3, Washington, DC, USA, 2004. IEEE Computer Society.

[13] Y. Labrou and T. Finin. Semantics and Conversations for an Agent Communication Language. In *Proc. of IJCAI-97*, pages 584–591, 1997.

[14] Q. Limbourg and J. Vanderdonckt. Comparing task models for user interface design. In D. Diaper and N. Stanton, editors, *The Handbook of Task Analysis for Human-Computer Interaction*, chapter 6. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 2003.

[15] H. Liu and M. Parashar. Accord: A Programming Framework for Autonomic Applications. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 36(3):341–352, May 2006.

[16] P. Luff, N. Gilbert, and D. Frohlich. *Computers and Conversation*. Academic Press, 1990.

[17] W. C. Mann and S. Thompson. Rhetorical Structure Theory: Toward a functional theory of text organization. In *Text*, pages 243–281, 1988.

[18] M. Nowostawski, D. Carter, S. Cranefield, and M. Purvis. Communicative acts and interaction protocols in a distributed information system. In *AAMAS '03: Proceedings of the second international joint conference on autonomous agents and multiagent systems*, pages 1082–1083, New York, NY, USA, 2003. ACM Press.

[19] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, England, 1969.